[正则 1](#_Toc99831018)

[1.1.1 分类 1](#_Toc99831019)

[1.1.2 作用 2](#_Toc99831020)

[1.2 \*元字符metacharacter 2](#_Toc99831021)

[1.2.1 \*普通字符 2](#_Toc99831022)

[1.2.2 \*单行模式和多行模式 4](#_Toc99831023)

[1.2.3 \*定位符 4](#_Toc99831024)

[1.2.4 ＊限定符 5](#_Toc99831025)

[1.2.5 \*特殊字符 5](#_Toc99831026)

[1.2.6 ＊选择 6](#_Toc99831027)

[1.2.7 \*引擎选项-修饰符 7](#_Toc99831028)

[1.3 \*贪婪与非贪婪 7](#_Toc99831029)

[1.4 \*反向引用 8](#_Toc99831030)

[1.5 \*运算符优先级 8](#_Toc99831031)

[1.6 \*匹配规则 9](#_Toc99831032)

[1.7 字符簇与确定重复出现 9](#_Toc99831033)

[1.8 练习 10](#_Toc99831034)

[1.9 Python正则 10](#_Toc99831035)

[1.9.1正则表达式修饰符--可选标志flag 11](#_Toc99831036)

[1.9.2单次匹配 11](#_Toc99831037)

[1.9.3全部匹配 12](#_Toc99831038)

[1.9.4匹配替换 12](#_Toc99831039)

[1.9.5分割字符串 13](#_Toc99831040)

[1.9.6编译 13](#_Toc99831041)

[1.9.7分组--必须是match对象 13](#_Toc99831042)

[1.10 参考链接： 13](#_Toc99831043)

# 正则

正则表达式文本处理极为重要的技术，它可对字符串按照某种规则进行检索、替换。

1970年代ken Thompson将正则表达式引入到unix中文本编辑器ed和grep命令中，使得正则表达式普及开来。在Perl语言扩展了很多新特性。

1997年Philip Hazel开发的PCRE： perl compatible regular expressions，被php和httpd等工具采用。

## 1.1.1 分类

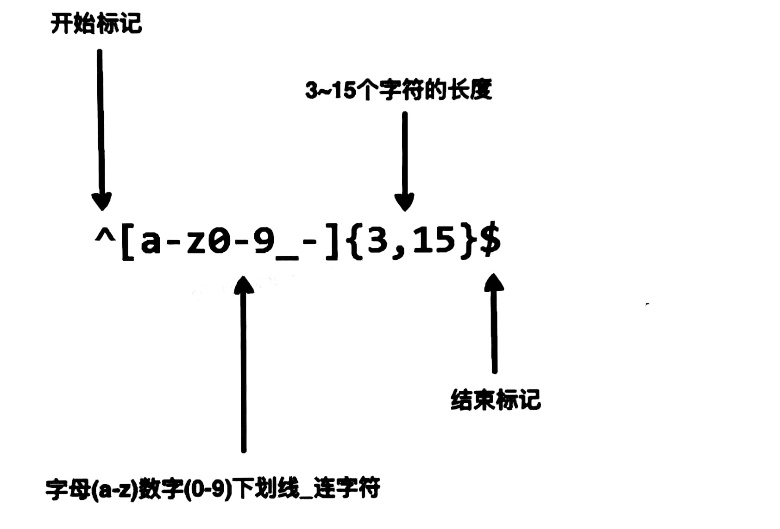
1、BRE：基本正则表达式：grep、sed、vi等

2、ERE：扩展正则表达式：egrep（grep-E）、sed-r等

3、PCRE：几乎所有高级语言都是PCRE的变种。python的SRE正则表达式引擎，可认为是PCRE的子集

## 1.1.2 作用

针对动态文本进行字符串测试（是否有手机、信用卡号）、搜索、替换、删除、提取子串。

例如：我们在写用户注册表单时，只允许用户名包含字符、数字、下划线和连接字符（-），并设置用户名的长度，我们就可以使用以下正则表达式来设定

# 1.2 \*元字符metacharacter

正则表达式（regular expression）描述了一种字符串匹配的模式（pattern），可以用来检查一个字符串是否含有某种子串、将匹配的子串替换或者从某个串中取出符合某个条件的子串等。

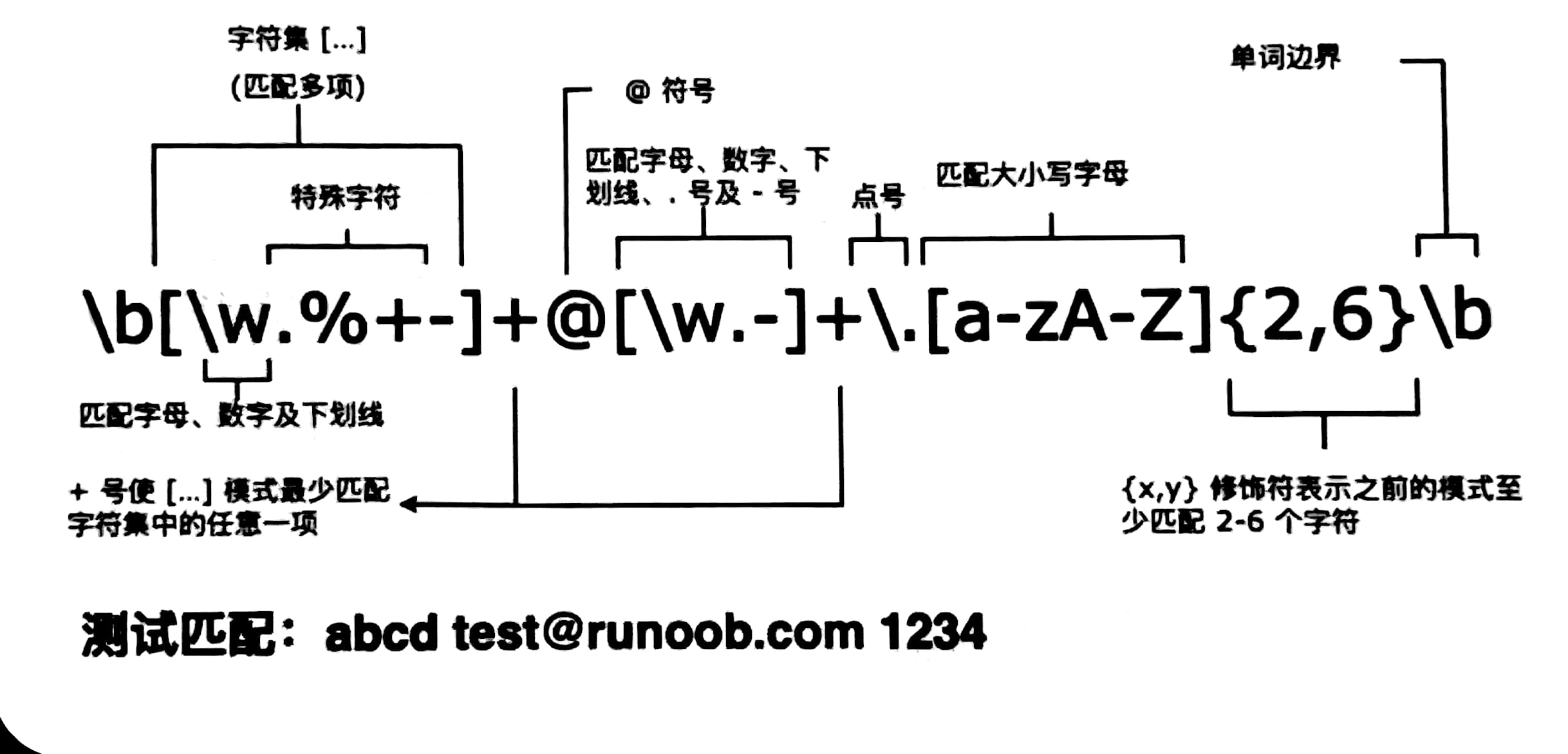
## 1.2.1 \*普通字符

|  |  |  |
| --- | --- | --- |
| 字符 | 描述 | 举例 |
| . | 单行模式：匹配除换行符以外的**任意一个**字符 多行模式：可以匹配所有字符，包括换行符 |  |
| [abc] | 字符集合，只能表示一个字符位置。匹配所包含的任意一个字符 | 匹配[…]中的所有字符，例如/[aeiou]/g 匹配字符串‘google runoob taobao’中所有的eoua字母。 |
| [^abc] | 字符集合，只能表示一个字符位置。匹配除去集合内字符的任意一个字符 | 匹配除了[…]中字符的所有字符，例如/[aeiou]/g 匹配字符串‘google runoob taobao’中除了aeiou字母的所有字母 |
| [a-z] | 字符范围，也是一个集合。表示一个字符位置。匹配所包含的任意一个字符 |  |
| [^a-z] | 字符范围，也是一个集合。表示一个字符位置。匹配除去集合内字符的任意一个字符 |  |
| \b | 匹配单词的边界。 | ＼bb在文本中找到单词中b开头的b字符 |
| \B | 不匹配单词的边界。 | t\B包含t的单词但是不以t结尾的t字符，例如write;\Bb不以b开头的含有b的单词，例如able |
| \d | ［0-9］匹配1位数字 |  |
| \D | ［^0-9］匹配1位非数字 |  |
| \s | 匹配1位空白字符，包括换行符、制表符、空格[\f\n\t\v] |  |
| \S | 匹配任何非空白字符。等价于 [^\f\n\t\v] |  |
| \w | 匹配［a-zA-Z0-9\_］,**包括中文的字** |  |
| \W | 匹配除\w以外的字符 |  |

实例：邮箱匹配, 测试匹配：abcd test＠runoob.com 1234

str = "abcd test@runoob.com 1234"

patt1 = ‘\b[\w.%+-]+@[\w.-]+\.[a-zA-Z]{2,6}\b’

re.match(patt1, str)

## 1.2.2 \*单行模式和多行模式

* 单行模式：可以匹配所有字符，包括换行符。

＾表示整个字符串的开头，$表示整个字符串的结尾

* 多行模式：可以匹配除换行符之外的字符。

^表示整个字符串的开始，$表示整个字符串的结尾。开始指的是\n后面紧接着的下一个字符；结尾指的是\n前的字符。（windows换行符\r\n,\n前的字符是\r，所有多行模式windows可能会出现问题：比如匹配以e结尾的字符串e$，字符串中看不见换行符\r\n，e＄只能匹配e\n；这个时候可以在结尾前加\r:＾\*\r$）

## 1.2.3 \*定位符

定位符能够将正则表达式固定到行首或行尾。它们还能够创建这样的正则表达式，这些正则表达式出现在一个单词内、在一个单词的开头或者一个单词的结尾。

定位符用来描述字符串或单词的边界,＾和$分别指字符串的开始与结束,\b描述单词的前或后边界,\B表示非单词边界。

|  |  |
| --- | --- |
| 字符 | 描述 |
| ^ | 匹配输入字符串开始的位置，整个字符串的开头。  开始指的是＼n后面紧接着的下一个字符 |
| $ | 匹配输入字符串结尾的位置。整个字符串的结尾。结尾指的是＼n前的字符。（windows换行符＼r＼n，＼n前的字符是＼r，所有多行模式windows可能会出现问题：比如匹配以e结尾的字符串e＄，字符串中看不见换行符＼r＼n，e$只能匹配e＼n；这个时候可以在结尾前加\r：＾.＊\r$） |
| \b | 匹配一个单词边界，即字与空格间的位置。 |
| \B | 非单词边界匹配。 |

## 1.2.4 \*限定符

限定符用来指定正则表达式的一个给定组件必须要出现多少次才能满足匹配。有\*或＋或？或｛n｝或｛n，｝或｛n，m｝共6种。

|  |  |
| --- | --- |
| 字符 | 描述 |
| \* | 匹配前面的子表达式零次或多次。例如，zo\*能匹配＂z＂ 以及＂zoO＂。\*等价于｛0，｝。 |
| + | 匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 ＂zo＂ 以及＂zoo＂，但不能匹配＂z＂。＋等价于 ｛1，｝。 |
| ? | 匹配前面的子表达式零次或一次。例如，＂do(es)?＂可以匹配＂do＂、 ＂does＂中的"does"、 ＂doxy＂中的＂do＂。?等价于｛0,1｝。 |
| {n} | n 是一个非负整数。匹配确定的n次。例如，‘o｛2｝’不能匹配＂Bob＂中的‘o’，但是能匹配 ＂food＂ 中的两个o。 |
| {n,} | {n,} n 是一个非负整数。至少匹配n 次。例如，‘o{2,}’不能匹配＂Bob＂中的‘o’，但能匹配＂foooood＂中的所有o。'o｛1,｝'等价于 ‘o+’。‘o｛0,｝’则等价于 ‘o\*’。 |
| ｛n, m｝ | m和n均为非负整数，其中n<=m。最少匹配n次且最多匹配m次。例如，＂o｛1,3｝＂ 将匹配 ＂fooooood＂ 中的前三个 o。'o｛0,1｝'等价于'o？'。请注意在逗号和两个数之间不能有空格。 |

## 1.2.5 \*特殊字符

要匹配这些特殊字符，必须首先使字符“转义＂，即，将反斜杠字符＼放在它们前面。

**（特殊字符在[]方括号内不用转义也可匹配）**

|  |  |
| --- | --- |
| (pattern) | 捕获或分组：使用小括号标记一个子表达式。捕获后自动分配组号从1开始，且可以获取供以后使用。  **注：括号中被捕获到的分组会被单独缓存起来，供使用，会占用内存，这是一个副作用。** |
| \数字 | 配合(pattern)使用，匹配对应的分组。  例如：(very)\1匹配very very，单捕获的组group是very |
| (?:pattern) | 仅改变优先级，不捕获分组，不缓存。  例如：industr(?:y|lies)，等价于industry|lindustries |
| (?:<name>exp）或者(?:'name'>exp) | 命名分组：命名分组捕获，可以通过name访问分组。  Python语法必须是（?P<name>exp）  不怎么用 |
| (?=exp) | 零宽断言：零宽度正预测先行断言 断言exp一定在匹配的右边出现，即：**断言后面一定跟一个exp**。 **注意：断言表达式exp只用作判断，不会被捕获，不会出现在最终匹配结果，不会占用分组号。** |
| (?!exp) | 负向零宽断言：零宽度负预测先行断言 **断言后面一定没有exp。** |
| (?<!exp) | 负向零宽断言：零宽度负回顾后发断言 断言前面一定没有exp前缀 |
| | | 指明两项之间的一个选择。要匹配|，请使用|。 例如：w|food，匹配w或food; (w|f)ood，匹配wood或food |
| (?#comment) | 注释：用作正则表达式中的注释，不用作任何匹配。少用 |

## 1.2.6 \*选择

用圆括号（）将所有选择项括起来，相邻的选择项之间用｜分隔。（）表示捕获分组，（）会把每个分组里的匹配的值保存起来，多个匹配值可以通过数字n来查看（n是一个数字，表示第n个捕获分组的内容,起始分组为1）。

但用圆括号会有一个副作用，使相关的匹配会被缓存，此时可用?:放在第一个选项前来消除这种副作用。其中?:是非捕获元之一，还有两个非捕获元是?=和?!，这两个还有更多的含义，前者为正向预查，在任何开始匹配圆括号内的正则表达式模式的位置来匹配搜索字符串，后者为负向预查，在任何开始不匹配该正则表达式模式的位置来匹配搜索字符串。

## 1.2.7 \*引擎选项-修饰符

标记也称为修饰符，正则表达式的标记用于指定额外的匹配策略。标记不写在正则表达式里，标记位于表达式之外，格式如下：／pattern／flags

例如：

匹配“hello,I’m from Num.4 middle school”

/^[a-z0-9,".\s]+$/i

|  |  |  |
| --- | --- | --- |
| 修饰符 | 含义 | 描述 |
| i | ignore-不区分大小写 | 将匹配设置为不区分大小写，搜索时不区分大小写：A和a没有区别。但是匹配结果仍然是原字符 |
| g | global-全局匹配 | 查找所有的匹配项 |
| m | multiline-多行匹配 | 使边界字符^和$匹配每一行的开头和结尾，记住是多行，而不是整个字符串的开头和结尾 |
| S | 特殊字符点.中包含换行符\n | 默认情况下.是匹配除换行符\n之外的任何字符，加上s修饰后，.中包含换行符 |

# 1.3 \*贪婪与非贪婪

**默认是贪婪模式，**即尽量多匹配更长的字符串。

**非贪婪：在重复的符号后面加一个问号？**

|  |  |
| --- | --- |
| \*? | 匹配任意次，但尽可能少重复 |
| +? | 匹配至少1次，但尽可能少重复 |
| {n, m}? | 匹配至少n次，至多m次，但尽可能少重复 |

例如：very very very happy使用v.\*y和v.\*?y：结果分别为very very very happy和very

# 1.4 \*反向引用

捕获组捕获到的内容，不仅可以在正则表达式外部通过程序进行引用，也可以在正则表达式内部进行引用，这种引用方式就是反向引用。反向引用的作用通常是用来查找或限定重复、查找或限定指定标识配对出现等等。

**对于普通捕获组和命名捕获组的引用，语法如下：**

* 普通捕获组反向引用：\k<number>，通常简写为\number
* 命名捕获组反向引用：\k<name>或者\k'name'

**反向引用必然要与捕获组一同使用的，**如果没有捕获组，而使用了反向引用的语法，不同语言的处理方式不一致，有的语言会抛异常，有的语言会当作普通的转义处理。

**举例说明**

源字符串：abcdebbcde

正则表达式：([ab])\1

对于正则表达式“([ab])\1”，捕获组中的子表达式“[ab]”虽然可以匹配“a”或者“b”，但是捕获组一旦匹配成功，反向引用的内容也就确定了。如果捕获组匹配到“a”，那么反向引用也就只能匹配“a”，同理，如果捕获组匹配到的是“b”，那么反向引用也就只能匹配“b”。由于后面反向引用“\1”的限制，要求必须是两个相同的字符，在这里也就是“aa”或者“bb”才能匹配成功。

考察一下这个正则表达式的匹配过程，在位置0处，由“([ab])”匹配“a”成功，将捕获的内容保存在编号为1的组中，然后把控制权交给“\1”，由于此时捕获组已记录了捕获内容为“a”，“\1”也就确定只有匹配到“a”才能匹配成功，这里显然不满足，“\1”匹配失败，由于没有可供回溯的状态，整个表达式在位置0处匹配失败。

正则引擎向前传动，在位置5之前，“([ab])”一直匹配失败。传动到位置5处时，，“([ab])”匹配到“b”，匹配成功，将捕获的内容保存在编号为1的组中，然后把控制权交给“\1”，由于此时捕获组已记录了捕获内容为“b”，“\1”也就确定只有匹配到“b”才能匹配成功，满足条件，“\1”匹配成功，整个表达式匹配成功，匹配结果为“bb”，匹配开始位置为5，结束位置为7。

扩展一下，正则表达式“([a-z])\1{2}”也就表达连续三个相同的小写字母，“([a-z])\1+”表示连续的n个相同的小写字母。

# 1.5 \*运算符优先级

正则表达式从左到右进行计算，并遵循优先级顺序，这与算术表达式非常类似。相同优先级的从左到右进行运算，不同优先级的运算先高后低。下表从最高到最低说明了各种正则表达式运算符的优先级顺序：

|  |  |
| --- | --- |
| 运算符 | 描述 |
| \ | 转义符 |
| (),(?:),(?=),[] | 圆括号和方括号 |
| \*,+,?,{n},{n,},{n,m} | 限定符 |
| ＾，＄，＼任何元字符、任何字符 | 定位点和序列（即：位置和顺序） |
| | | 替换，“或”操作。字符具有高于替换运算符的优先级，使得＂m|food＂匹配＂m＂或＂food＂。若要匹配＂mood＂或＂food＂，请使用括号创建子表达式＂(m|f)ood＂。  **注意｜具有最低优先级：** **例如：匹配IP地址不超过255，'^[1-9]?\d[1-2][1-5][1-5]$'将得不到预期结果。因为表达式匹配的是＾[1-9]?d或[1-2][1-5][1-5]$,**  **应该写为:'^[1-9]?\d$|^[1-2][1-5][1-5]$'，或通过圆括号改变优先级：'^([1-9]?\d|[1-2][1-5][1-5])$'** |

# 1.6 \*匹配规则

* 字符＾和$同时使用时，表示精确匹配（字符串与模式一样）。
* 如果一个模式不包括＾和$，那么它与任何包含该模式的字符串匹配：

例如：once 与“There once was a man from NewYork “Who kept all of his cash in a bucket.”是匹配的

# 1.7 字符簇与确定重复出现

字符簇[]：把一些字符放在方括号[]里，表示其中的任意一个字符

确定重复出{}：跟在字符或字符簇后面的花括号{}用来确定前面的内容的重复出现的次数。

# 1.8 练习

* 匹配3位数
* 2、匹配ip地址
* 3、选出含ftp的链接，且文件类型是gz或xz的文件名

**# 优先级测试**

If re.match('(w|f)ood', 'woodfood'):

print(re.match('w|food', 'woodfood').group(0))

**# 匹配任意三位数**

’^([1-9]\d\d?|\d)\r?$’

po = '^([1-9]\d\d?|\d)\r?$'

if re.match(po,'999'):

print(True)

**# 匹配ip地址**

＃ip地址规则：四段，每段1-3个数字，每段不超过255

p1 = "^(?:(?:[1-9]?\d|[1-2][1-5][1-5]).){3}(?:[1-9]?\d|[1-2][1-5][1-5])$"

if re.match(p1,'2.255.255.25'):

print('IP ok')

＃使用socket库：

print('ip:',socket.inet\_aton('2.255.255.255'))

**# 选择ftp链接，且文件类型是gz或xz**

p2='^ftp:.\*\.(?:gz|xz)$'

p3='(?<=.\*ftp.\*/)[^/].(?:gz|xz)'

# 1.9 Python正则

re模块：用来匹配字符串（动态、模糊的匹配），爬虫用的多

## 1.9.1正则表达式修饰符--可选标志flag

re.I：使匹配对大小写不敏感

re.L：做本地化识别（locale-aware）匹配

re.M：多行匹配，影响＾和＄

re.S：使.匹配包括换行在内的所有字符

re.U：根据Unicode字符集解析字符。这个标志影响\w,\W,\b,\B.

re.X：该标志通过给予你更灵活的格式以便你将正则表达式写得更易于理解。

## 1.9.2单次匹配

* re.match(pattern，string，flags＝0)：尝试从字符串的起始位置匹配一个模式，匹配成功re.match方法**返回一个匹配的对象，只匹配一次**，如果不是起始位置匹配成功的话，match()就返回none。用group()，来获取匹配结果，返回字符串，例如：

line = "Cats are smarter than dogs"

#.\*表示任意匹配除换行符(\n、\r)之外的任何单个或多个字符

#(.\*？)表示“非贪婪”模式，只保存第一个匹配到的子串

matchObj = re.match( r'(.\*) are (.\*?) .\*', line,re.M|re.I)

if matchObj:

print("matchObj.group():",matchObj.group())

print ("matchObj.group(1) :",matchObj.group(1))

print ("matchObj.group(2) :", matchObj.group(2))

else:

print("No match!!")

* re.search(pattern，string，flags＝0)：扫描整个字符串并返回第一个成功的匹配。匹配成功re.search方法返回一个匹配的对象，匹配失败返回none。用group()，来获取匹配结果，返回字符串。例如：a＝＂123abc456＂

print(type(re.search("(?P<value>[0-9]\*)([a-z]\*)([0-9]\*)",a).group()))

* re.fullmatch(pattern，string，flags＝0)：全长匹配，整个字符串和正则表达式匹配

start() 返回匹配开始的位置

end() 返回匹配结束的位置

span() 返回一个元组包含匹配（开始,结束）的位置

## 1.9.3全部匹配

* re.findall(pattern, string, flags=0)：在字符串中找到正则表达式所匹配的所有子串，并**返回一个列表**，如果没有找到匹配的，则返回空列表
* re.finditer(pattern, string, flags=0)：和findall类 似，在字符串中找到正则表达式所匹配的所有子串，并把它们作为一个迭代器返回。

## 1.9.4匹配替换

* re.sub(pattern, repl, string, count=0,flags=0)：用于替换字符串中的匹配项。返回替换后的字符串

repl：替换的字符串，也可为一个函数，例：

＃将匹配的数字乘于2

def double(matched):

value = int(matched.group('value'))

return str(value\*2)

s='A23G4HFD567'

print(re.sub('(?P<value>\d+)', double, s))

？P＜value＞代表的是为group分组添加一个分组名，为group命令，全要在前面添加一个问号，如果问号没有添加，就没有命名成group。

count：模式匹配后替换的最大次数，默认0表示替换所有的匹配

* re.subn(pattern, repl, string, count=0, flags=0):用于替换字符串中的匹配项。返回一个元组，（替换后的字符串，替换次数）

## 1.9.5分割字符串

* re.split(pattern，string[，maxsplit＝0，flags＝0])：分割.按照能够匹配的子串将字符串分割后返回列表.

re.split（separator，[separator]）：可以指定多个分隔符对字符串进行分割：

text＝＇你好！吃早饭了吗？再见。＇

print(re.split('。|!|?',text))

## 1.9.6编译

* re.compile(pattern[，flags])：用于编译正则表达式，生成一个正则表达式（Pattern）对象，供match（）和search（）这两个函数使用。例如：

pattern = re.compile（r＇\d＋＇） ＃ 用于匹配至少一个数字

m=pattern.match（＇one12twothree34four＇）＃查找头部，没有匹配

## 1.9.7分组--必须是match对象

使用小括号的pattern捕获的数据被放到了组group中。

match和search函数可以返回match对象；findall返 回字符串列表；finditer返回一个个match对象。

如果pattern中使用了分组，如果有匹配的结果，**会在match对象中**：

* 使用group(N)，返回对应的分组，1-N是对应的分组，0返回整个匹配的字符串
* 如果使用了命名分组，可以使用group(＇name＇)的方式取分组
* 使用groups()返回所有分组的tuple：m.groups()==(m.group(0),m.group(1),...)
* 使用groupdict()，返回所有命名分组

# **1.10 参考链接**

[正则表达式 – 语法 | 菜鸟教程 (runoob.com)](https://www.runoob.com/regexp/regexp-syntax.html)

[Python3 正则表达式 | 菜鸟教程 (runoob.com)](https://www.runoob.com/python3/python3-reg-expressions.html)

[正则表达式之反向引用 - gq\_orange - 博客园 (cnblogs.com)](https://www.cnblogs.com/guorange/p/6693168.html)